

AUTOMATED RATING REVIEW SYSTEM

1.PROJECT OVERVIEW

This project predicts customer review ratings based on textual feedback. It involves cleaning and preprocessing review text, balancing the dataset to avoid class bias, and transforming text into TF-IDF features. The system uses a stratified train-test split to ensure fair evaluation and prepares the data for machine learning models, providing insights into review patterns and customer sentiment.


2. ENVIRONMENTAL SETUP

- Python: 3.9+
- Libraries: pandas, numpy, nltk, scikit-learn, re
- NLTK Resources: stopwords, punkt, wordnet
- IDE: Google Colab and VS Code
- Hardware: Laptop

3.GITHUB PROJECT SETUP

Created GitHub repository : **automated-review-rating-system**


Structure of directory









 **automated-review-rating-system** Public

Pin Watch

main 1 Branch 0 Tags

Add file <> Code

 **RiginmReji** Add modular code and preprocessing scripts 52c08ad · yesterday 24 Commits

 app	Add modular code and preprocessing scripts	yesterday
 data	Added .gitkeep files to make folders visible	2 weeks ago
 frontend	Add modular code and preprocessing scripts	yesterday
 models	Add modular code and preprocessing scripts	yesterday
 notebooks	Added .gitkeep files to make folders visible	2 weeks ago
 review_env	Add modular code and preprocessing scripts	yesterday
 README.md	Add clean project README	yesterday
 requirements.txt	Add modular code and preprocessing scripts	yesterday

4.DATA COLLECTION

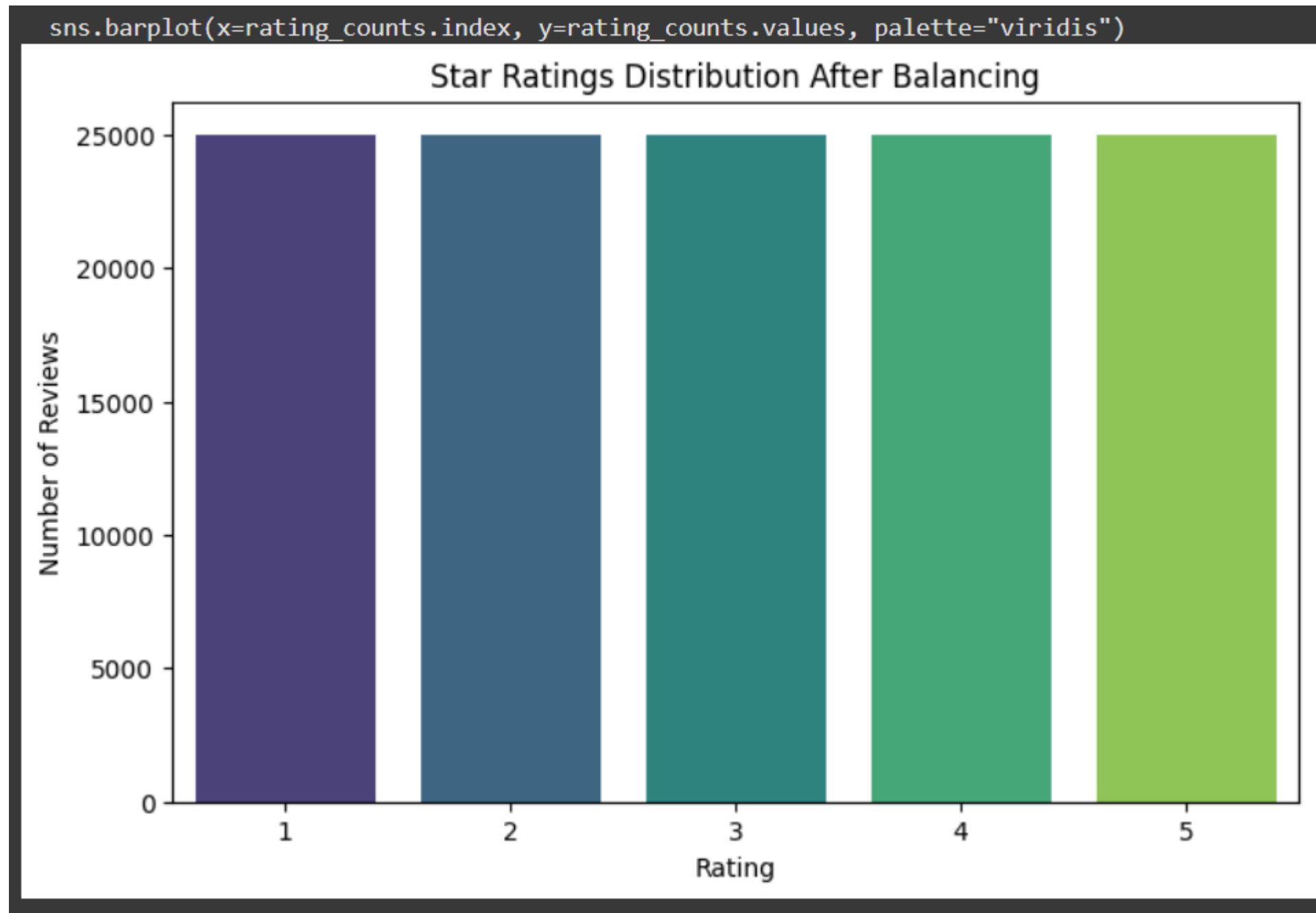
- The dataset was collected from Kaggle, containing real-world customer reviews from e-commerce platforms.
- It has 568454 rows and 10 columns
- Dataset link = [https://www.kaggle.com/snap/amazon-fine-food-reviews/downloads/amazon-fine-](https://www.kaggle.com/snap/amazon-fine-food-reviews/downloads/amazon-fine-food-reviews.zip)
- [food-reviews.zip](https://www.kaggle.com/snap/amazon-fine-food-reviews/downloads/amazon-fine-food-reviews.zip)

It includes the review text and corresponding star ratings (1–5), along with optional metadata such as product IDs and timestamps

- Initial preprocessing involved removing duplicates, handling missing values, and filtering extremely short or long reviews to ensure quality for analysis and modeling.
- Final dataset contains 2 columns with Rating and Review_text
- 1 star - 52234, 2 star - 29739, 3 star - 42606, 4 star - 80546, 5 star - 363007

4.1 .Balanced dataset

- Link for balanced dataset = [Upload files · RiginmReji/automated-review-rating-system](#)
- Balanced dataset was created with 25000 rows of each rows
- Bar chart to visualize the distribution of ratings in the dataset



5.DATA PREPROCESSING

Effective data preprocessing is essential to improve the performance and accuracy of machine learning models. The following techniques were applied to prepare the raw review dataset for modeling.

5.1. Removing duplicates

Duplicate rows containing the exact same review and rating were removed to prevent bias and overfitting. This ensured that the dataset only included unique observations.

Code:

```
before = review.shape[0]
review = review.drop_duplicates(subset=['Review_text'])
after = review.shape[0]

print(f"Removed {before - after} duplicate reviews.")
print(f"Remaining reviews: {after}")
```

5.2. Removing conflicting reviews

Some reviews had identical text but different star ratings. These inconsistencies can confuse the model. Such conflicting entries were identified and removed to maintain label clarity.

Code:

```
conflicts = review.groupby("Review_text")["Rating"].nunique()  
conflicting_reviews = conflicts[conflicts > 1]  
  
print("Number of conflicting reviews:", conflicting_reviews.shape[0])
```

5.3 Handling missing values

Rows with missing or null values particularly in the review text or rating column - were removed. This step ensured the dataset was complete and meaningful for analysis.

Code:

```
print(review.isnull().sum())
```

```
review = review.dropna(subset=["Review_text"])
```

```
review = review[review["Review_text"].str.strip() != ""]
```

```
print("Shape after handling missing values:", review.shape)
```


5.4 Dropping unnecessary columns

Non-essential columns such as user IDs, product id, summary, HelpfulnessNumerator were dropped. These fields did not contribute to the model and could introduce noise or privacy concerns.

5.5 Lowercasing Text

All review text was converted to lowercase to maintain uniformity. This helps prevent duplication of tokens like "Good" and "good" being treated as separate words.

5.6 Removing URLs

URLs present in the review text were removed using regular expressions.

5.7 Removing emojis and Special characters

Emojis and special symbols may not contribute meaningful context for text analysis (unless specifically required). We remove these characters to focus on the core textual content.

5.8 Removing punctuations

Emojis and special symbols may not contribute meaningful context for text analysis (unless specifically required). We remove these characters to focus on the core textual content.

5.9 Stopwords Removal

Stopwords are commonly used words (like the, is, and, to, in) that usually carry little meaningful information in text analysis. Removing stopwords helps reduce noise and improves the performance of text-based models.

Example:

Before stopword removal:

"This is a very good product and I like

it a lot" After stopword removal:

"This very good product like lot"

Code:

```
import nltk
from nltk.corpus import stopwords

nltk.download("stopwords")
stop_words = set(stopwords.words("english"))

review["Review_text"] = review["Review_text"].apply(
    lambda x: " ".join([word for word in x.split() if word.lower() not in stop_words])
)

print("After stopwords removal:")
print(review["Review_text"].head(10))
```

5.10 Lemmatization

Lemmatization is the process of converting words into their base or dictionary form, known as a lemma. Unlike stemming (which just chops word endings), lemmatization considers the context and part of speech, producing meaningful root words.

Example::

- “running” → “run”
- “better” → “good”
- “cars” → “car”

why lemmatization is better than stemming ?

Meaningful Output:

Stemming just chops off word endings, which may produce non-words.

- “studies” → “studi”
- Lemmatization uses a dictionary to return valid base words.

“studies” → “study”

- Stemming = speed, Lemmatization = correctness.

5.11 Filtering by wordcount

Very short reviews might not contain enough context to be useful, and excessively long reviews could be outliers. We apply filtering to exclude:

- Reviews with fewer than 3 words
- Reviews that exceed 300 words

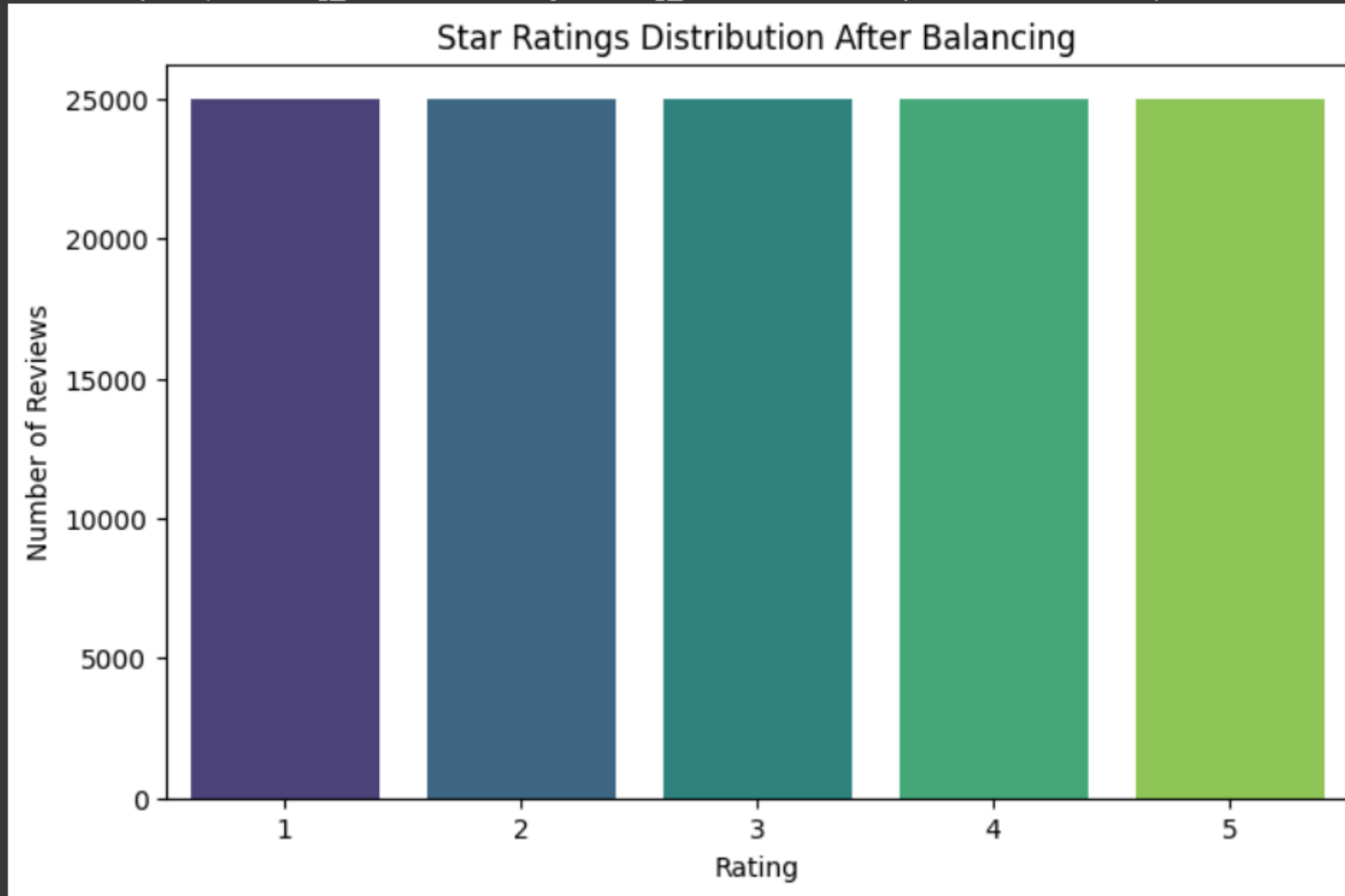
This ensure that the dataset remains robust and relevant for model training

6. Data visualisation

Box Plot

Box Plot is a visualization that shows the spread and distribution of review lengths across ratings in the dataset. It highlights the median review length, the range of most reviews, and detects outliers such as unusually short or long reviews. This helps in understanding how review size varies with customer ratings.

```
sns.barplot(x=rating_counts.index, y=rating_counts.values, palette="viridis")
```



1.sample of 1

star rating

showing 3 samples

for rating 1

1. product arrived labeled jumbo salted peanuts the peanuts actually small sized unsalted
sure error vendor intended represent product jumbo
2. cats happily eating felidae platinum two years got new bag shape food different tried new
food first put bowls bowls sit full kitties touch food ive noticed similar reviews related formula
changes past unfortunately need find new food cats eat
3. candy red flavor plan chewy would never buy

2.sample of 2 star rating

1. looking secret ingredient robitussin believe found got addition root beer extract ordered
good made cherry soda flavor medicinal

2. love eating good watching tv looking movies sweet like transfer zip lock baggie stay fresh
taketime eating
3. purchased mango flavor doesnt take like mango hint sweetness unfortunately hint
aftertaste almost like licorice ive consuming various sports nutrition products decades im familiar
come like taste products ive tried mango flavor one least appealing ive tasted terrible bad
enough notice bad taste every sip tak

3.sample of 3 star rating

1. seems little wholesome supermarket brands somewhat mushy doesnt quite much flavor
either didnt pass muster kids probably wont buy
2. flavors good however see difference oaker oats brand mushy
3. stuff buy big box stores nothing healthy carbs sugars save money get something least taste

4.sample of 4 star rating

1. confection around centuries light pillowy citrus gelatin nuts case filberts cut tiny squares
liberally coated powdered sugar tiny mouthful heaven chewy flavorful highly recommend yummy

treat familiar story cs lewis lion witch wardrobe treat seduces edmund selling brother sisters witch

2. got wild hair taffy ordered five pound bag taffy enjoyable many flavors watermelon root beer melon peppermint grape etc complaint bit much redblack licoriceflavored pieces particular favorites kids husband lasted two weeks would recommend brand taffy delightful treat
3. good flavor came securely packed fresh delicious love twizzlers

5.sample of 5 star rating

1. bought several vitality canned dog food products found good quality product looks like stewprocessed meat smells better labrador finicky appreciates product better
2. great taffy great price wide assortment yummy taffy delivery quick taffy lover deal
3. saltwater taffy great flavors soft chewy candy individually wrapped well none candies stucktogether happen expensive version fralingers would highly recommend candy served beachthemed party everyone loved

7. Train-Test-Split

Train-Test Split is a technique to divide the dataset into two separate sets:

- Training Set (typically 80%): Used to train the machine learning model.
- Test Set (typically 20%): Used to evaluate the model's performance on unseen data.

This separation ensures that the model generalizes well and is not just memorizing the training data.

Why stratified split?

In classification problems like review rating prediction, stratified splitting is important to maintain class balance (equal distribution of ratings) in both training and testing sets.

Without stratification, some rating classes (like 1-star or 5-star) may be underrepresented in the test set, leading to biased evaluation.

How it was done?

To prepare the data for model training, the dataset was first shuffled randomly to eliminate any order bias. A stratified train-test split was then performed using `train_test_split()` from `sklearn.model_selection` with the `stratify=y` argument to ensure that all star ratings were proportionally represented in both training and testing sets. The dataset was split using an

80% training and 20% testing ratio. After splitting, all text preprocessing steps —including lowercasing, lemmatization, stopword removal, and cleaning — were applied separately to X_train and X_test to prevent data leakage and maintain model integrity.

code:

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

def prepare_train_test(df, text_col="Review_text", target_col="Rating",
                      test_size=0.2, max_features=25000):
```

```
    """
```

```
    Perform stratified train-test split and TF-IDF vectorization.
```

```
    Args:
```

```
        df (pd.DataFrame): Input dataframe (already balanced).
```

```
        text_col (str): Column with text data.
```

```
        target_col (str): Target label column.
```

```
        test_size (float): Proportion of test split.
```

max_features (int): Number of TF-IDF features.

Returns:

```
    X_train, X_test, y_train, y_test, vectorizer
"""

# Train-test split (stratified ensures equal class balance in both sets)
X = df[text_col]
y = df[target_col]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=test_size, stratify=y, random_state=42
)

# TF-IDF Vectorization
vectorizer = TfidfVectorizer(max_features=max_features)
X_train_vec = vectorizer.fit_transform(X_train) # fit only on training data
X_test_vec = vectorizer.transform(X_test) # only transform test data

return X_train_vec, X_test_vec, y_train, y_test, vectorizer
```

8.Text Vectorization

Machine learning models can't work directly with raw text —they require numerical input. Vectorization is the process of converting text data into numerical features.

TF-IDF (Term Frequency - Inverse Document Frequency)

TE-IDE is a statistical technique that represents how important a word is to a document relative to the entire corpus.

- TF (Term Frequency): How often a word appears in a document.
- IDF (Inverse Document Frequency): Penalizes common words and highlights rare but important ones.

This approach helps to capture both word relevance and discriminative power, making it better than simple word counts.

FORMULA FOR TF-IDF

$$\text{TF-IDF}(t,d,D) = \text{TF}(t,d) \times \text{IDF}(t,D)$$

In words:

- **TF** gives local importance (within a document).
- **IDF** reduces the weight of common words (like “the”, “and”).
- Their product (**TF-IDF**) balances both.

code :

```
from sklearn.feature_extraction.text import TfidfVectorizer  
from sklearn.model_selection import train_test_split
```

```
X = review["Review_text"] # cleaned reviews  
y = review["Rating"]      # target labels
```

```
X_train, X_test, y_train, y_test = train_test_split(  
    X, y, test_size=0.2, random_state=42, stratify=y  
)
```

```
tfidf = TfidfVectorizer(  
    max_features=25000, # use top 25000 words for efficiency  
    ngram_range=(1,2), # include unigrams + bigrams  
    stop_words="english" # remove common stopwords  
)
```

```
X_train_tfidf = tfidf.fit_transform(X_train)  
X_test_tfidf = tfidf.transform(X_test)  
print("TF-IDF Train shape:", X_train_tfidf.shape)  
print("TF-IDF Test shape:", X_test_tfidf.shape)
```